

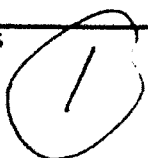

AD-A265 306



DOCUMENTATION PAGE

Form Approved

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 1992		3. REPORT TYPE AND DATES COVERED THESIS XXXXXX	
4. TITLE AND SUBTITLE Development and Implementation of a General Purpose X-Windows Display Program				5. FUNDING NUMBERS 	
6. AUTHOR(S) Captain Gregory D. Carson					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT Student Attending: University of Iowa				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/CI/CIA-92-128	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFIT/CI Wright-Patterson AFB OH 45433-6583				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release IAW 190-1 Distribution Unlimited ERNEST A. HAYGOOD, Captain, USAF Executive Officer				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) <div data-bbox="264 1344 660 1617" data-label="Text"> <p>DTIC ELECTE JUN 10 1993</p> </div> <div data-bbox="908 1407 1205 1533" data-label="Text"> <p>93-12638</p>  </div>					
14. SUBJECT TERMS				15. NUMBER OF PAGES 78	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT		18. SECURITY CLASSIFICATION OF THIS PAGE		19. SECURITY CLASSIFICATION OF ABSTRACT	
				20. LIMITATION OF ABSTRACT	

DEVELOPMENT AND IMPLEMENTATION OF
A GENERAL PURPOSE X-WINDOWS IMAGE DISPLAY PROGRAM

by
Gregory Donald Carson

Accession For		
NTIS	CRA&I	<input checked="" type="checkbox"/>
DTIC	TAB	<input type="checkbox"/>
Unannounced		<input type="checkbox"/>
Justification		
By		
Distribution		
Availability Codes		
Dist	Availability or Special	
A-1		

A thesis submitted in partial fulfillment
of the requirements for the Master of
Science degree in Biomedical Engineering
in the Graduate College of
The University of Iowa

December 1992

Thesis supervisor: Professor Edwin L. Dove

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Gregory Donald Carson

has been approved by the Examining Committee
for the thesis requirement for the Master of
Science degree in Biomedical Engineering at the
December 1992 graduation.

Thesis committee: Edw. J. Dine
Thesis supervisor

Thelma S.
Member

IRL
Member

ACKNOWLEDGEMENTS

A special thanks to David Koblas, the author of xpaint, whose programming examples and code were used with permission to create a shell for the program developed here. And my deepest appreciation for the assistance and patience of my wife, Marian, without whom none of this would have even been possible.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vi
INTRODUCTION	1
 CHAPTER	
I. THE X WINDOWS SYSTEM	3
Why Use the X Windows System	3
How the X Windows System is Organized	7
II. WHAT THE SEP PROGRAM DOES	13
What is the SEP Program	13
Opening and Initialization	14
Handling Events	22
SEP Overview	25
SEP Functions	29
The Draw Function	29
The Menu Bar Function	30
The Save Function	31
The Quit Function	32
The Erase Function	33
The Images Function	33
III. INTERFACING WITH SEP	35
What is Required of the SEP Interface	35
The C Language Interface	36
Starting a New Process	37
The Interprocess Communication Channel ...	38
The FORTRAN Language Interface	41
Inter Language Data Addressing	43
IV. USING THE SEP PROGRAM	45
Introduction	45
Getting the SEP Program Running	46
Starting the Stand Alone Version	46
Starting SEP from the FORTRAN Language ...	47
Starting SEP from the C Language	50
Using SEP	52
Manipulating Images	57
The SEP Image Format	57
Converting to the SEP Image Format	57

Inverting the Colors of an SEP Image	59
Getting Hard Copy	61
V. TESTING AND EVALUATION OF SEP	62
CONCLUSION	67
Completing the Package	67
Enhancing SEP	68
REFERENCES	70

LIST OF TABLES

Table		Page
1.	Test Image Sizes and Results	63
2.	Test Image Pixel Comparisons	64

LIST OF FIGURES

Figure	Page
1. Block Diagram of X Windows System	10
2. Diagram of the SEP Program Window Placement	17
3. The SEP Program Window with Sample Image After Opening and Initialization	22
4. Flowchart for Overall SEP Program	26
5. Flowchart for SEP Events	27
6. The SEP Pop-Up Windows	31
7. Process Fork Example Code Fragment	38
8. Opening a Communication Channel with Pipes Programming Code Fragment	39
9. Interprocess Communication with Fork and Pipe Diagram	40
10. FORTRAN Syntax to Call the SEP Program	47
11. FORTRAN Language Compile Command Example	48
12. Example FORTRAN Language Program Using the SEP Program	49
13. C Language Syntax to Call the SEP Program	50
14. C Language Compile Command Example	51
15. Example C Language Program Using the SEP Program	51
16. SEP Menu and Pop-Up Windows	55
17. Example Program to Convert to SEP Image Format ..	58
18. Example Program to Invert Colormap	60

INTRODUCTION

The purpose of the work presented here was to design and develop a portable, easy-to-use software package capable of displaying a gray scale image. The display of an image was in actuality only a part of the work. The program was also required to be both FORTRAN and C callable. Additionally, the program had to provide the user with the capability to draw on the image in color, and import tracings, or contours, and display them on the image. The final requirement was to engage recursive use of the program, to provide the ability to display multiple images simultaneously.

This work is a presentation of the program developed to meet the above stated requirements. The program is called SEP. The first chapter contains a brief discussion of the reasoning behind the choice of the windowing system utilized. Also presented is an overview of the terminology and concepts involved in using that windowing system. The second chapter describes the internal operations of the program and how it meets the requirements. The third chapter is a description of the method used to interface the developed program to other computer programs. The fourth chapter then provides an explanation of how to use the developed program. The final

chapter is a discussion of the testing and evaluation done to ensure the program meets its goal.

CHAPTER I

THE X WINDOWS SYSTEM

Why Use the X Windows System

The development of a graphical computer program involves many things. The first of these things is the choice of environments. For the purpose of developing this graphical program, the term environment can be defined as the windowing system in which it will operate. Thus, the first step toward the completion of this program was choosing which windowing system to use.

Windowing systems provide the user with a much simpler interface to the computer and its programs. They also provide the programmer with a much simpler means to write graphics-based programs by taking care of some of the display details. Windowing systems are typically pointer and menu driven. Some of the more common examples of windowing systems include the Apple Macintosh Operating System, and the Microsoft Windows program package for Microsoft Disk Operating System (MS DOS) compatible computer systems. The big advantage of a windowing system is that it makes the use of programs much simpler for the user. Such systems tend to minimize the need for specific knowledge of the internal

workings of computer programs. Basically windowing systems make computers more human literate.

So which is the best windowing system to use? This is a difficult question because it involves determining which of the available windowing systems is the standard. In the world of computers and computer programming the answer to this question is frequently based on opinion and experience. Those systems most familiar to the programmer or those with which she/he has achieved the most success for her/his purposes are likely to be the standard to her/him. While this is a complication when choosing a computer standard, it does not prove to be such a large problem in the case of windowing systems. While there is some subjectivity to the statement, "this windowing system is the standard", that very statement can almost be made about the X Windows System.

The X Windows System, or X, is basically a mechanism for displaying device-independent bit-mapped graphics through a network-transparent windowing system [5]. "Device-independent" refers to a given programmer's freedom from hardware constraints. "Network transparency" means that application programs can be run on machines scattered throughout the connected network. This statement alone serves to point out one of the major advantages of the X Windows System: Portability. The X Windows System is highly portable. Portability refers to the ability to use a program under different computer operating systems. The more

portable a program is the easier it is to compile and execute under different operating systems.

When developing computer programs that are intended for widespread use and distribution the subject of program portability becomes an issue rather quickly. Portability of computer programs is a genuine concern with the wide variety of computer manufacturers and operating systems available today. Similarly, when developing a graphical computer program the windowing environment becomes an important portability concern. If a program is not portable then it's intended use becomes improbable because it would be too difficult to re-write the program for each new system.

After the search for a windowing system in which to develop the image display program, only X seemed to have the widespread use and portability desired. The X Windows System is considered by many to be an industry standard [13]. X Windows is a software system that provides a large set of primitives, or basic operations, that can be combined as desired to develop a graphical computer program. This flexibility is also an important consideration when choosing an environment in which to develop graphical software. X Window's intentional lack of a specific graphical user interface provides the advantage of freedom in program design and implementation. The freedom of flexibility does have a cost. In this case that cost is complexity. The X Windows System is an extremely complex system to use. It remains a

fortunate fact, however, that the most portable windowing system is also very flexible and very customizable.

While the portability of X windows may be debatable it is probably the only windowing system that runs on every major operating system and computer platform built today [5]. While this may be hard to prove, there is some evidence which suggests this to be the case.

The X Windows System can be run with equal ease on all workstation operating systems that are currently in production: UNIX, VAX/VMS, Sun OS, Next, Dec, etceteras. In fact, even the Macintosh and MS DOS operating systems can run X Windows. The X Windows System, although not 'public domain', is available for free and can be redistributed without fee. This fact has greatly contributed to its widespread availability and use.

The X windows system was developed by the MIT X Consortium to become a standard. This goal in itself does not make it a standard, or even prove that it could become one. But, nearly any experienced programmer can attest to the fact that the Consortium has had an impact. Even a brief scan through usenet news can also be rather convincing. There are many boards devoted to the development of both the X Windows System and programs that use the system.

What the future holds for computers, windowing systems and other software will likely remain in a constant state of change. Therefore, the programmer today is forced to make a

decision as to what the 'standard' is and is likely to be, and develop software accordingly. For the reasons presented here, the decision was made to use the X Window System; thus began the development of an image display program.

How the X Windows System is Organized

The X Windows System is a complex system. Though its complexity does not detract from its value as a highly portable open-ended windowing system, X Windows uses a lot of unique terminology and concepts. It may therefore be prudent to present some of the more common terms and points of organization.

In X, a display is defined as a workstation with a keyboard, mouse, or other pointing device, and a monitor [7]. This definition implies X was designed with computer networks in mind. In fact, the X windows system is network based. Part of the power of X comes from its network based orientation. This fact gives rise to some important terminology. X includes the capability of running programs on machines other than the display in front of which the user is placed. Each display is controlled by a special program. The program that controls each display is called the server [7].

The server functions as an intermediary between user programs and the display. User programs are called clients [5]. The server passes user input to the clients. User

input is keyboard entries and pointer movements. It also handles the various network messages to and from clients. The server is responsible for access to the display by clients. The server also stores the many complex data structures required by X. These data structures are known as resources and include graphics contexts (GC), colormaps, fonts, etc. Finally, the server is responsible for two dimensional drawing. Here two dimensional drawing means graphics are performed by the display server rather than the client. Thus, the client must give drawing instructions for its windows, text, etc to the server.

This relationship between the clients and the server means clients must be able to communicate with the server. In X these communications are referred to as events. Events are things that happen to clients, between clients, and include user input and interactions between clients and the server. A simple example of interactions between clients would be the movement of windows. If one clients window is on top of another, and the first one is moved to expose the second, an event would let each program know what had happened. The clients in this example would be responsible for directing the appropriate redrawing of the visible parts of their windows.

Moving windows occurs rather often in X, as do many other similar situations, such as resizing of windows, starting new ones, etc. One might wonder if there is so much

window interaction going on, how does the server have time to do anything but keep up with events? As it turns out there is a special type of client called a window manager which takes care of most of these interactions. The window manager is the only client with this server type privilege. The window manager is typically the first client started on a display. Its job is to handle the window interactions and to provide a graphical user interface (GUI).

A GUI has four components [1]. First, it performs the basic text and drawing functions. Second, it provides a means of determining input focus, (i.e. which of the windows of the screen is to be the active window). Third, it provides a tool kit which enables programmers to write programs that make use of the sub-routines of the window system. And fourth, the GUI provides a style guide that ensures a fairly consistent look (appearance) and feel (behavior) to the clients active on the display.

Figure 1 is a diagram of an X Window based system with some clients running. Note that each display typically has its own X server running. This is not required but is normally the case.

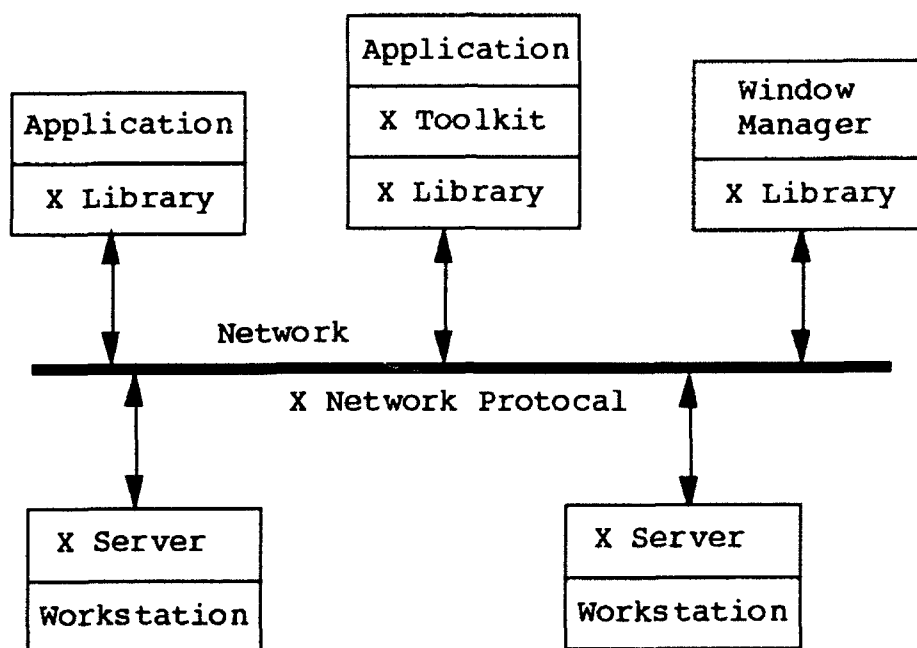


Figure 1: Block Diagram of X Windows System

From the figure it can be seen that there is still more to the X Windows system. Notice in the figure that the network connection between displays also has the label X Network Protocol, and that under each client is the phrase X Library. Clients communicate with the server and the window manager by sub-routine calls to a low-level library of C language routines called the Xlib [5]. Xlib, the X Library as shown in Figure 1, is a large collection of functions. These functions include opening a connection to a display, creating various types of windows, drawing graphics, responding to events, and a myriad of other things. Xlib actually communicates via the X network protocol.

Because of the large volume of calls necessary to use the Xlib, the X Windows System often includes a second library. This second library, which functions as in Figure 1, exists essentially on top of Xlib, and is called the XtIntrinsics, or X Toolkit. The X Toolkit, as with other toolkits such as those provided by the window manager, is an object-oriented library of widgets [7]. These widgets can be called to perform pre-defined sets of Xlib calls to meet a specific goal. Thus, the obvious limitation of tool-kits is that calls to their routines can only perform their predetermined functions. This fact indicates that use of a tool-kit is not as flexible as direct use of the Xlib calls.

One final note regarding terminology. Any Xlib call or structure will have an X at the beginning of its name. For example, the XOpenDisplay function. The calls to the X Toolkit will have an Xt at the beginning. For example, the XtInitialize function.

Thus, X Windows programs, while immensely powerful in the world of windowing systems, are by nature extremely complex. X programs must deal with window managers, their relationship to the server and the display, with events, as well as perform their required tasks. The nature of SEP led to the decision to maintain maximum control within the environment, and thus it was written primarily with the Xlib. The reasons for this decision will be detailed in the next chapter.

For a complete description of Xlib calls, refer to either Nye [5 & 6], Scheifler [10] or on-line documentation. For a complete description of Xtoolkit calls, reference either Nye [7 & 8] or on-line documentation.

CHAPTER II

WHAT THE SEP PROGRAM DOES

What is the SEP Program

This chapter will explain what the SEP program does internally to display an image and create the drawing, contour importing, and saving functions. This is not a description of how to use the program; rather, it is a description of the mechanics of the functions contained within the SEP program, and how they come to display the image and perform the other functions. This chapter will also include a description of the X events that are handled by the program and how they are dealt with.

Any X windows program, to be functional, must do at least two things. Those two things are: (1) create its windows; (2) handle the events that are presented to it by other clients and the server. This is the first part of what the program developed here does. The program developed is called the SEP program. The name is arbitrary and has no official meaning.

The SEP program is an X Windows program written primarily with the Xlib. It displays an image and creates tools for drawing on the image and importing contours. It

also includes a function for saving the displayed image. The SEP program is comprised of many sub-programs, each with many sub-routines. The entry into the program is through a routine which calls other routines to open and initialize the windows, set up the environment, and handle the events that occur.

Opening and Initialization

The first task addressed is to open the display. Opening the display is the terminology used by X windows for opening a connection to the workstation. This action provides a pointer to the display which is used by the program when making the call to create windows, event handlers, and other actions needed by the program to perform its functions. The display is opened by a call to the Xlib function `XOpenDisplay`.

After the display is open to the program, it then begins to set up the organization of its windows. It creates several windows, all through calls to the `XCreateWindow` function. The parent window, or the master window of the SEP program, is created first. The parent window is the window in which all the other windows created by SEP are placed. The image display window, or draw window, is the second window created. This is followed in turn, by the menu bar window, and the pop-up windows for the quit, save, erase, and

images functions. These functions are described in the fourth section of this chapter.

These windows are placed on the parent in a specified location so they are not overlapping and therefore always remain visible. The pop-up windows are an exception to this rule. They are placed on top of the other children windows when called in order to get the users attention. The positioning action is accomplished by creating a position variable that is dependent upon which window is being created, its size, and desired placement. Thus, each child window's position is related to the position of the other windows. Forcing the sub-windows, or children, of the parent to be in specific locations ensures proper visibility of all created windows. In SEP it serves to ensure the image and action window (the menu bar window) are completely and properly visible for the user.

The opening and initialization calls for all of the windows include minimum sizing information and placement information from the position variable. Placement information for the parent window is provided by filling in the XSizeHints structure. The XSizeHints data structure is one of the resources stored on the display. This allows the SEP program to determine the parent window's size and placement. Filling in the XSizeHints structure also provides an easy mechanism for passing size information to the children. The children windows then use the size and

position variable information to establish data for their specified sizes and positions as related to each other within the parent. Thus, if the SEP program window is moved or resized, all of the children windows will remain in their proper places, and the image will remain displayed properly.

Sizing information is based on input from the image. Specifically, the size of the image determines the minimum size of the SEP parent window. Each of the children windows is then in turn based on the image size. All children windows are placed in the same orientation regardless of the size of the image, up to the maximum declared image size (1024 by 1024 pixels). In general terms, the "draw window", or the image display window, is centered on the parent. The menu bar window is placed across the top of the SEP window. Reference the diagram of the SEP window below in Figure 2. Figure 2 also shows the cursor position information. This data is displayed, by the draw function, on the SEP parent window in the lower left corner. The cursor position information function is discussed in the fourth section of this chapter along with the draw function.

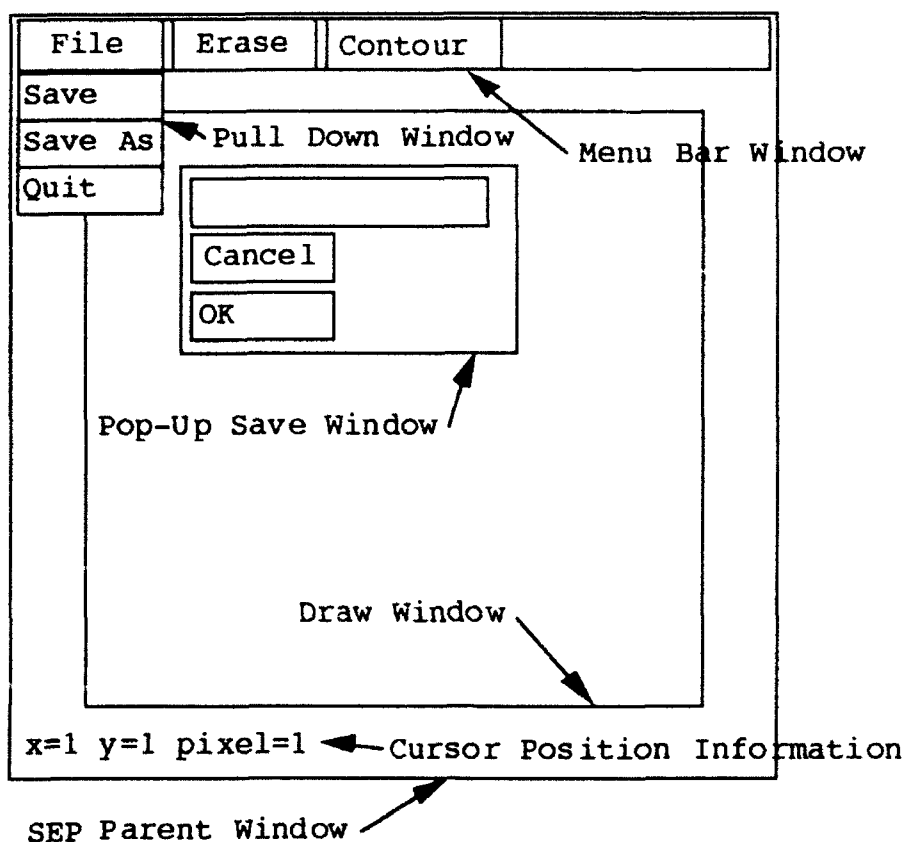


Figure 2: Diagram of the SEP Program Window Placement

The menu bar window contains many sub windows. These are the children of the menu bar window. The children of the menu bar window are those which contain the pull down menus, and the various pop up windows. An example of one set of pull-down windows and a pop-up window, the save window, is also shown in Figure 2. These children are not shown initially along with the draw and menu bar windows. It is necessary, however, to create these windows in the initialization process so that they can be mapped rapidly

when needed. "Mapping" is X terminology for actually placing or drawing the windows on the display monitor so they are visible to the user.

In addition to the creation of the necessary windows, SEP also sets up several graphics contexts (GC's), creates its own colormap, and allocates memory space for images and data. A GC is a data structure that contains information needed for drawing; such as the width of lines, the foreground and background colors, the fonts to be used for displaying text and so on. It is stored in the workstation memory to speed accomplishment of these tasks. A colormap is really nothing more than a table used to determine the colors associated with image values.

The SEP program creates a colormap by storing the pixel value data returned from the XAllocNamedColor and XAllocColor functions in an internal array variable. This array variable then becomes the colormap that SEP will use for displaying pixels. SEP requires 137 colors be available on the machine it will be run on. There are 9 colors for drawing and importing contour data, and 128 colors for a grayscale from black to white. The program is intended to be used for displaying grayscale images with pixel value data ranging from 0 for black to 255 for white.

The use of 128 colors to satisfy this grayscale range was required in order to avoid conflicts with the common window managers. Window managers require a number of colors

be allocated to them, typically this number is between 30 and 50. Since the majority of X capable workstations available today support only one 256 position colormap at a time, it was necessary for the SEP program to utilize less than this number.

This quantization of the grayscale from 256 positions to 128 may have the effect of 'smearing' the edges of an image. While the normal user cannot distinguish between any two adjacent colors in a 256 position grayscale [3] the cumulative effect may slightly 'smear' the edges present in an image. This 'smearing' effect may be diminished by dithering the image.

SEP stores its colormap data in an internally held array variable, with 256 positions. The first nine colors, or colors 0 to 8, are determined by the return values from the XAllocNamedColor function. This function is given the name of a color, and returns the color data structure for the closest hardware supported color available on the display. These first colors are reserved for drawing and display of imported contours.

The remaining colors, from 9 through 255, are determined by repeated calls to the XAllocColor function. Grayscale color values are generated with equal parts for each of the red, green, and blue color contributions. This data is then used by the XAllocColor function to obtain the color data structure for the closest color the hardware can support.

Again, the pixel value part of this data structure is stored in the array variable. Since 128 colors are being generated and placed in 247 positions, all but the first and last three colors are stored in two adjacent colormap positions.

If the display running the SEP program has a grayscale monitor, the first nine colors will be various shades of gray. If the display has a color capable monitor these colors will be fairly close to the shades each of blue, green, and red requested.

The array variable data resulting from the above process contains 256 pixel value pointers. They are arranged in intensity order from 9 to 255, for the grayscale, and 0 to 8 for the drawing colors. The SEP program then uses this array as a set of pointers to displayable colors for each pixel value in a grayscale image.

The use of these low-end color values, or colors, is not considered a significant decrease in the available gray scale for two reasons. First, using 9 of 256 color values is less than four percent of the gray scale. Second, and more importantly, the color values in both the high end, those close to 256, and the low end, those close to 0, are in most cases not seperably distinguishable from colors close to them in value. In fact, it is quite likely that these values are outside of the dynamic range of the monitor and will appear identical to the user.

Once the colormap has been created, the various GC's used by the program are created. The GC's are used to set the foreground and background colors of all SEP windows, as well as to set the font to be used for the display of text in the menu bar windows. The XCreateGC function is called to set this information into memory so it is available as it is needed.

After the size, position, inter-relationships, colormap, and GC's are created and stored, the image to be displayed needs to be created. The image is passed into the SEP program as an array of data points. The first step in the display of the image is to allocate memory to hold the image information. Memory is allocated, its addressing structure is defined, and the image array information is placed in it. To display the image, SEP calls the XCreateImage function to create an X Windows image. An X image is created and stored in memory to speed up redisplay actions. The program then calls the XPutPixel function to place each image data point into the X Windows image. Once all data points have been placed in the X Windows image the image is flushed to, or put in, the SEP display window. This is done through a call to the XPutImage function.

The final step in the open and initialization process is to map the windows. After the image has been drawn into the draw window, and the mapping has occurred, the appearance of the SEP program window is as below in Figure 3. The image

shown here is an example of an ultrasound image, digitized from a video tape. Virtually any image that can be converted into a simple integer data array can be displayed by the SEP program. (See chapter 4)

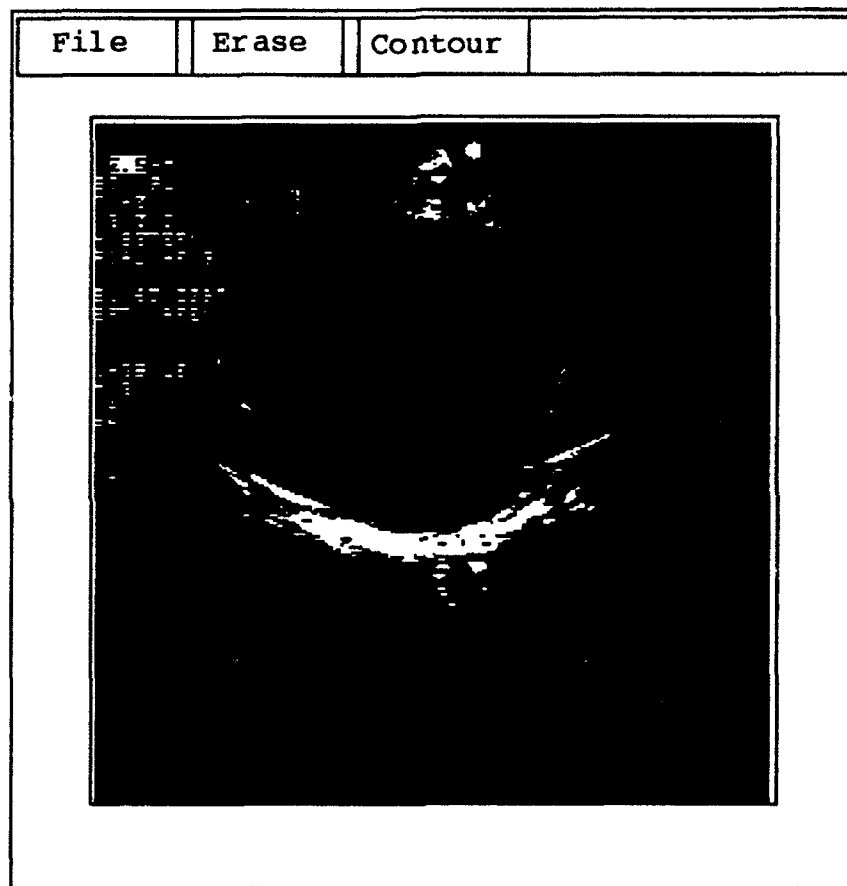


Figure 3: The SEP Program Window with Sample Image After Opening and Initialization

Handling Events

While the window environment of SEP was being opened and initialized; the windows created, the image placed in the

display window, and the initial windows mapped; the program was also declaring which events were to be addressed for each of the windows. An event is a message sent by the X server to the client when something has changed. A few examples of events include the movement of the mouse, the changing of a windows position, and the resizing of a window.

Every stable X program must establish which kinds of events it will handle. A program establishes the events it will handle by requesting the information from the server. The server passes event information in packets called event masks. Event masks contain the event data necessary for the windows routines to respond. A program, then, must define the event masks it will need for each window. This action is essentially identical to defining the event kinds a window will handle. The SEP program windows respond to several kinds of events. Those events are the Expose, KeyPress, ButtonPress, and PointerMovement events.

Expose events are those events that require the redrawing of windows. The best way to understand Expose events is to review a few examples. Movement of a window from one place to another on the screen causes an Expose event. The affected program must redraw its window(s) in the new location. Lowering or raising a program's windows also causes an Expose event. The program must redraw the portion(s) of its windows that are exposed to the user. The last example is resizing. When a program's windows are

resized the program must redraw its windows accordingly. All SEP program windows respond to expose Events in this manner.

The next type of event is the KeyPress event. A KeyPress event is X windows response to an entry on the displays keyboard. Some of SEP's windows respond to KeyPress events by drawing the corresponding character to the screen in a particular place, and the other simply ignore them. There is one special KeyPress event that all SEP windows respond to, that is the interrupt. In the case of SEP interrupt has been defined as control (^) Q, ^U, or ^C. All SEP windows respond to this KeyPress event by un-mapping their windows. Additionally the SEP program will terminate in response to the interrupt KeyPress event.

ButtonPress events are also handled differently by the SEP windows. A ButtonPress event occurs in response to the pressing of one of the buttons of the pointer. Linked to this event is another event called a ButtonRelease event. This is the corresponding release of the pointer button. Several of SEP's windows respond only to the combination of a ButtonPress and ButtonRelease event. The various windows reactions to this kind of event is described below.

The last kind of event is the PointerMovement event. This event occurs each time the displays pointer is moved to a new pixel position on the screen. Only some of SEP's windows perform a task in response to this kind of event, the other simply ignore it.

In X programs all events are handled by a subroutine called the event handler. SEP's event handler is a routine with several subroutines, one for each major function, or window. This structure permits each window to perform individual responses to only those events that it requires to conduct its function.

SEP Overview

The SEP program is comprised of several functions. These functions are the draw, menu, save, quit, erase, and images functions. The program code follows the flowchart shown in Figure 4 below. The program starts with the opening and initialization process which includes the creation of all windows for the program. Refer to the flowchart and the first section of this chapter for a complete description of this process.

Once all the windows have been created and mapped execution control is passed to the event handler. The event handler follows the flowchart found in Figure 5 below. This is where execution control is passed between the various functions of SEP. Refer to section two of this chapter and the flowchart for a more complete description of the event handler.

In short, the draw function allows pointer controlled drawing on the displayed image, and creation of the cursor position information text. The menu, or menu bar, function

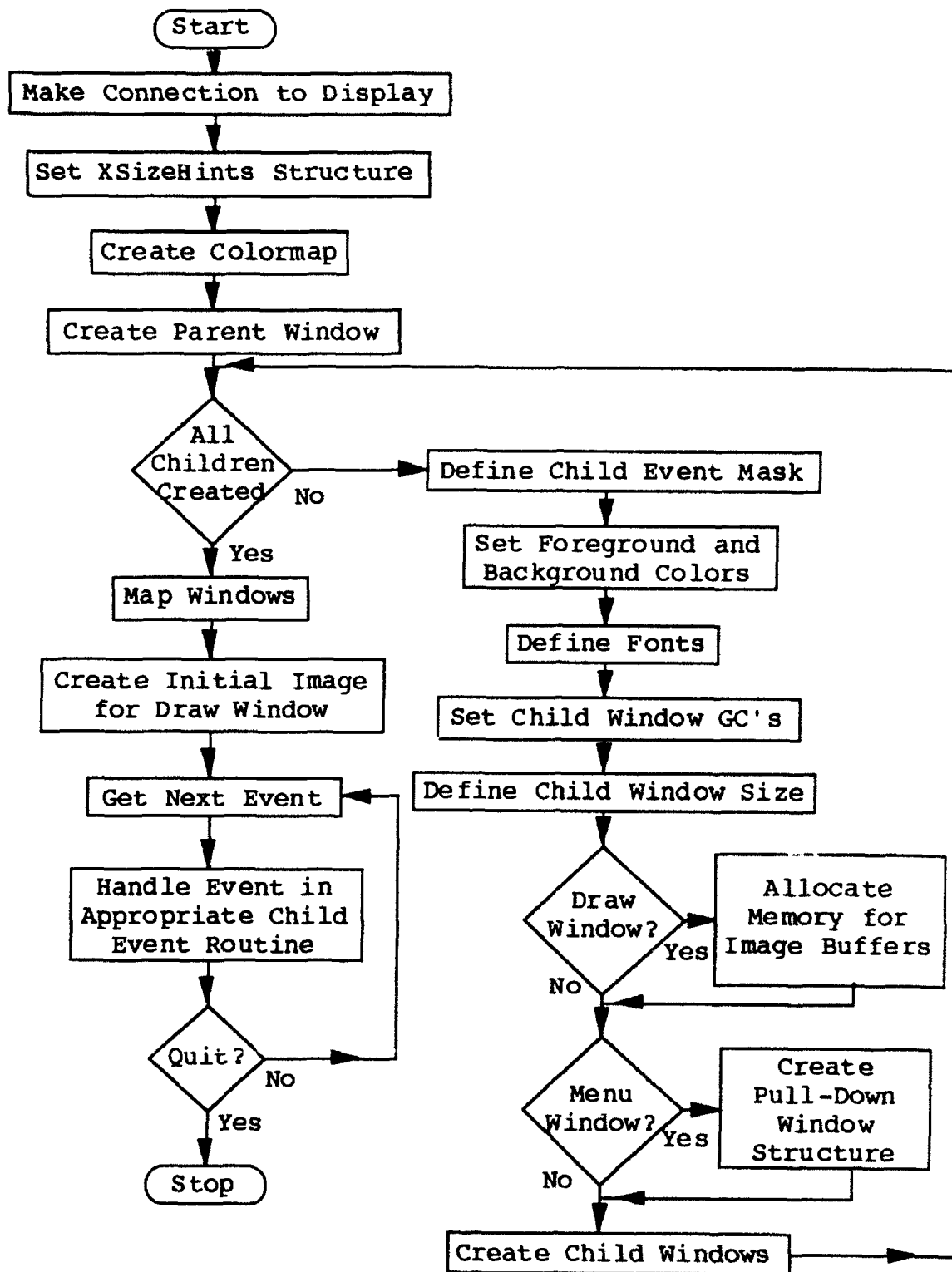


Figure 4: Flowchart for Overall SEP Program

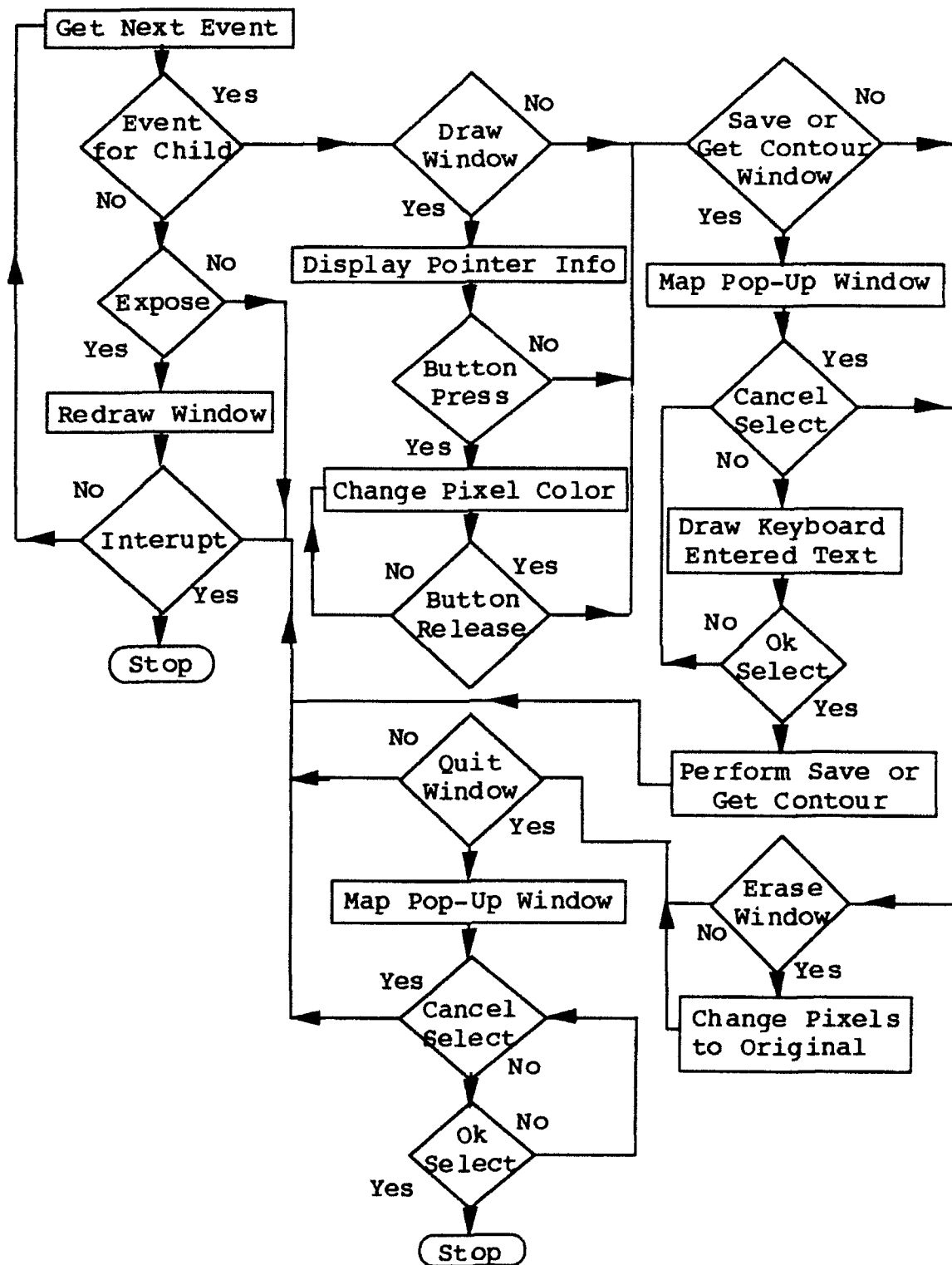


Figure 5: Flowchart for SEP Events

provides a set of pull down menus that access the remainder of the functions. The save function allows the user to save the displayed image. The quit function does just that, it terminates program execution. The erase function erases drawing on the image. And, finally, the images function provides for importing contour and new image data into the draw window.

As it can be seen from the event flowchart, figure 5, as each event is processed the program also looks for an interrupt. SEP responds to three common interrupts, the control (^) Q, the ^U, and the ^C. Entry of any of these characters will cause the program to stop.

One last point of note involves the versions of the SEP program. There are two versions of the program. First is the stand alone version. The stand alone version functions as a program unto itself. The second version is the 'callable' version. This version provides a means for FORTRAN and C language programs to display images and input new data. The second version displays an image and contains all the same functions as the stand alone version with the addition of a communication channel. The communication channel is used by the calling program to send additional data to the SEP program pixel by pixel for display. This is discussed in detail in chapter 4.

SEP Functions

As it has been mentioned the SEP program is comprised of several functions. The first of these, in order of creation, is the draw function. The other functions are the menu bar, save, quit, erase, and images functions.

The Draw Function

The draw window is the place where the image is displayed. The draw function uses three kinds of events, they are the Expose, ButtonPress and PointerMovement events. If the draw function is passed a Expose event it will redraw its window and the image appropriately. The real business of the draw function is to allow drawing on the image with the pointer. If the draw function receives a ButtonPress event it begins and continues changing the color of the pixel the pointer is on until it a ButtonRelease event is received. The new color of the pixel is one of the first nine colors, the current version is set to color two from the array. This part of the draw function only affects the draw window.

The draw function also draws the text for the cursor position information onto the SEP parent window. The information is retrieved from the PointerMovement event and the image data array. It is drawn onto the SEP parent window directly as text with no additional window.

The Menu Bar Function

The menu bar window and its function are the point of origin for all of the remaining SEP functions. The menu bar function is a pull down menu that passes execution control to those other functions. This function uses the Expose, ButtonPress, and PointerMovement events. As with all SEP functions having windows the Expose event is used to redraw windows as is necessary.

If the menu bar function is passed a ButtonPress event, and the event occurred on one of the menu bar items (File, Erase, or Images) that items pull-down menu is mapped, see Figure 6. As the pointer is moved down the pull-down menu the item the pointer is on will be highlighted. If a ButtonRelease event occurs on an item in the pull-down menu this function will pass execution control to that function. The new function selected will execute and then return control to the event handler. Just before execution control is passed the menu bar function will un-map the pull-down windows displayed.

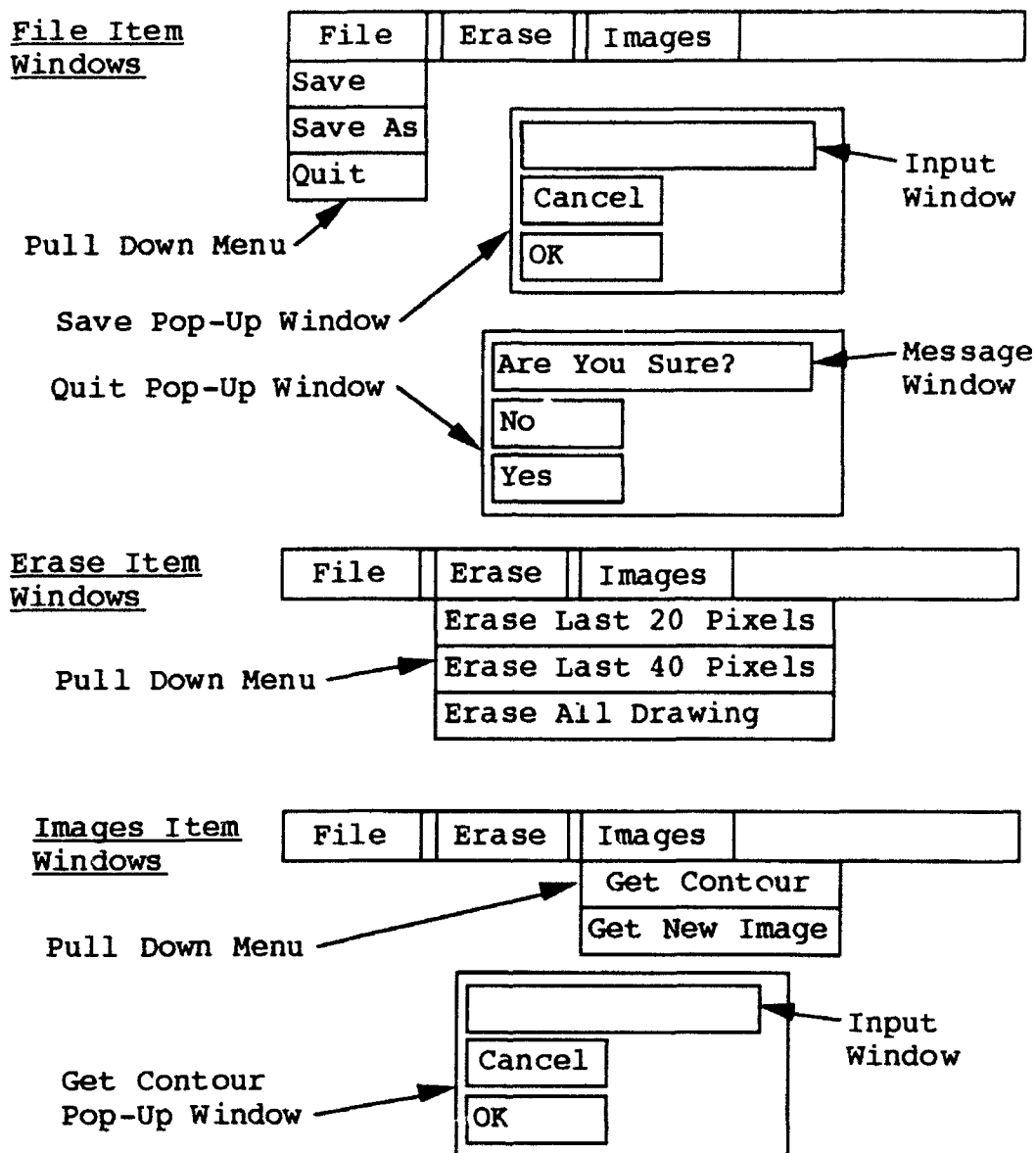


Figure 6: The SEP Pop-Up Windows

The Save Function

The save function begins by mapping a pop-up window, see Figure 6. This function responds to Expose events like all other SEP windows. It also responds to KeyPress, and

KeyPress events. If the save function receives a KeyPress event it will draw the corresponding character in the Input Window, refer to Figure 6. If this function receives a ButtonPress-ButtonRelease event combination in its 'Cancel' window, the function will terminate and return control to the event handler. If the save function gets a ButtonPress-ButtonRelease event combination in its 'Ok' window, it will save the image currently displayed in the draw window using the text displayed in the Input Window as the filename. Once this accomplished execution control will be returned to the event handler. Note, the save function saves images in the SEP file format which is discussed in Chapter 4, section 3. This function will also save contour data at the same time it is saving the image, again the file format for the contour data can be found in chapter 4.

The Quit Function

The quit function begins by mapping a pop-up window, refer to Figure 6. This function responds to Expose and ButtonPress events. The quit function redraws its windows as necessary in response to being passed an Expose event. The quit function will display a message in its 'Message Window' depending on the status of the drawing done on the image. If there is no unsaved drawing on the image the question "Are You Sure?" will be displayed. If unsaved drawing is present on the image a warning will be displayed.

The quit function responds to the ButtonPress-ButtonRelease combination similar to how the save function does. If the combination is received in the quit functions 'No' window control will be returned to the event handler. If the combination is received in the 'Yes' window program execution will be terminated and all windows unmapped.

The Erase Function

The erase function does not respond to events. This is because it has no windows. The erase function will restore the pervious image data in the increment selected from the menu bar pull-down menu, see Figure 6. Either 20, 40, or all drawn on pixels will be restored to the colors that were displayed before drawing occurred. Once this is accomplished control will be returned to the event handler.

The Images Function

The images function begins with a pop-up window identical to the pop-up window for the save function, see Figure 6. This function also responds to the same events and in the same manner as the save function with one exception. This function does not save to the character string drawn in the 'Input Window', rather it will attempt to get either the contour or image with that as a filename depending on the item selected from the menu bar pull-down window. If the function is not successful in opening the file, or the file

is in the wrong data format, a warning will be issued to the terminal window SEP was called from and control will be passed back to the event handler. If the file is opened the data will be drawn into the draw window, and then control will be returned to the event handler.

CHAPTER III

INTERFACING WITH SEP

As stated previously, there are two versions of the SEP program, the Stand Alone version, and the Callable version. The interface to the Stand Alone version is simple, because in essence there isn't one. It is simply started from the command line and utilized as described in Chapter IV. The callable version, on the other hand, requires some significant interfacing to utilize.

What is Required of the SEP Interface

The SEP program was developed as an image display program, which was meant to be used by other programs that generate images or perform image processing. Previous to the development of this program a hardware solution was in use. That hardware system is the DeAnza display system. Because this hardware system is usable, or is callable, from both the FORTRAN and C programming languages it was necessary to provide these functions to the SEP program as well.

Additionally, it was necessary for the program to be able to return control to the calling program so that the image processing program could continue its work while the

image display was being processed. This requirement made it necessary to establish a communication channel between the image processing program and the SEP image display program.

This chapter describes the mechanics of the routines written to create an inter-process communication channel between an image processing program and the SEP program. It includes a description of the interface to the SEP program from both the FORTRAN and C computer languages. It should be noted that the program written to handle the interprocess communication and execution of a separate process was written in the C language for both language interfaces.

The C Language Interface

The SEP program was written in the C language. The reason for this is simple. The X Windows Xlib was written in C, and was intended to be a C-callable library. Thus the interface of the SEP program to the C language was essentially already built into the SEP program itself. However, due to the additional requirements of returning control to the calling program and interprocess communication, it was necessary to write an interface program for the C language.

The interface program which allowed the starting of the SEP program and returned control to the calling program performs two functions. Those functions are to fork, or spawn, a separate process from the image processing program's

current, and create an open communication channel between the two processes. The communication channel is used to pass additional image data to be displayed from the image processing program to the SEP display program. The initial image array data is passed to SEP by sending a pointer to a memory stored array.

Starting a New Process

Under the UNIX operating system there are several means of starting one program by another program, and several means of creating a communication channel between two separately executing programs. The first issue to be addressed is starting a second program in a separate process, or spawning a process.

The two primary methods of starting a program from another program, under UNIX, are the fork and exec* system calls. The fork system call creates a duplicate process of the one from which it was called. The exec* system call executes a program by name. The interface program written to allow an image processing program to start the SEP program actually uses both. The fork system call is used to create a new process, in which the exec* system call is used to execute the SEP program.

This process is simple. The interface program simply contains calls to the system functions fork and exec*. An example of this is shown below in Figure 7.

```

        for(i=0;i<1;i++){
            if(!fork())
                execl(SEP());
        }

```

Figure 7: Process Fork Example Code Fragment

The example shows the fork call being used to call the exec system call. Using this method, the SEP program was started in a process separate from the process that is running the image processing program. In the actual code utilized, the exec* system call is also used to pass several pieces of information to the SEP program. That information is passed as if it were command line arrangements. In this case, information about the data string to be read from the interprocess communication channel, and the name of the memory file for the initial image data is passed. This information is simply included in the exec call. For example, to pass two pointers to data called ptr1 and ptr2, the execl line in the example shown in Figure 7 would be changed to execl ("SEP", ptr1, ptr2, 0);.

The Interprocess Communication Channel

In the programming of an interprocess communication channel there is really only one method feasible under the UNIX operating system. Reference is made here to the use of sockets for interprocess communication. Sockets, however, are not UNIX standard, rather they are part of the BSD

version of UNIX. In the case of this program, as it turns out, it was also much simpler to open this communication channel using pipes. This is because forks and pipes are intimately related [13].

The establishment of pipes was accomplished using the pipe system call. An example of this process is shown in Figure 8, below. It should be noted that in UNIX when starting a process from another process, the original process is called the parent, and the new process is called the child by convention.

```

if((pid = fork()) == 0)
{
    /* fork success */
    close(Pipes[1]); /* We don't need the write
end in the child */
    sprintf(buf, "%d", Pipes[0]);
    if(execl("SEP", "SEP", buf, (char *)0) == -1)
    { perror("execl() failed");
      exit(1);
    }
    exit(0);
}

if( pid == -1 ) /* fork unsuccessful */
{ perror( "fork() failed" );
  exit(1);
}

close(Pipes[0]); /* We don't need the read end
in the parent */

```

Figure 8: Opening a Communication Channel with Pipes Programming Code Fragment

The pipe system call is used to create the pipes. This system call places in a provided array two file descriptors, one for reading, and one for writing. These descriptors are connected "back-to-back". The pipes must be established before the fork occurs so that the creation of the new process with the fork call can carry with it the file

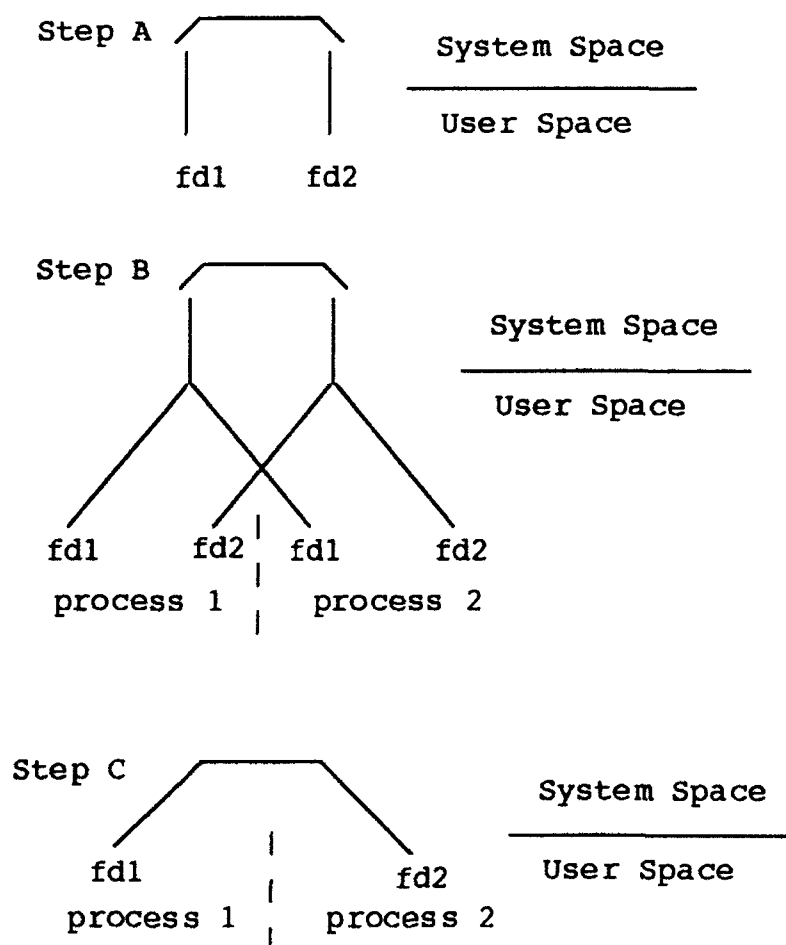


Figure 9: Interprocess Communication with Fork and Pipe Diagram

descriptors for the pipe. Figure 9 shows diagrammatically how the two functions fork and pipe are related, and what is done to open a one-way interprocess communication channel.

The pipe call returns two file descriptors, one read and one write, as shown in Figure 9, Step A. When the fork call is made all the file descriptors are duplicated and still connected to the same pipe, as shown in Figure 9, step B. To ensure that the communication channel is one way, allowing the calling program to send data to the called program, both parent and child must close one of the file descriptors. In this case, the image processing program will be left with the write side of the pipe, and the SEP program will be left with the read side of the pipe as in Figure 9, Step C.

One C language routine was written to both create the pipes and execute the SEP program in a separate process. It is this routine that is called by a C language image processing program when image display is desired. The actual interface method for a C language program to SEP is identical to calling any other C language routine from another C language routine.

The FORTRAN Language Interface

The interface of a FORTRAN program to the SEP program, as mentioned earlier, is through the C routine described above that spawns a new process and opens an interprocess communication channel. The two languages, FORTRAN and C,

have different naming conventions in their compiler interfaces. The FORTRAN language on the Icaen Apollo workstation network has two interfaces for its compiler, the ftn and f77 interfaces. Reference Apollo Domain System documentation for a complete description of both.

Unfortunately, the two compiler interfaces provided on the Apollo machines do not use the same naming conventions for program names. Since apart from this difference the two interfaces are nearly identical, one interface was selected. The f77 interface was selected. Thus the interface of the FORTRAN language to the SEP programs described here is only valid for the f77 FORTRAN compiler interface. The f77 interface is compliant with the ANSI standard for FORTRAN 77, and therefore should maintain the SEP programs portability.

Using the f77 interface provided a rather simple inter language calling mechanism. When a FORTRAN program needs to call a C program, it can do so in the usual manner with one minor naming change. The called C program must have an underscore character immediately following the name. For example, if the C program to be called is named 'sep', to be FORTRAN callable it must be renamed 'sep_'. A FORTRAN program then would use the name 'sep', to call this C program.

The calling of a C program from a FORTRAN program does however have a limitation. The C routine called can only be passed a single variable pointer. Thus to pass an image

array, and the width and height data, from the FORTRAN program to a C program some reorganization of data needs to be done. In this case the solution is fairly simple. The FORTRAN program stores the width and height data in the first two pixel positions of the image array. The C program then simply reads the width and height data from these pixel positions and uses the rest of the data as the image array data.

InterLanguage Data Addressing

There is one additional point concerning the interlanguage routine calling and data transfer between FORTRAN and C. As it turns out, the two languages address information in data arrays differently.

The FORTRAN language address data in a column-major ordering. This means that the elements of a two-dimensional array are stored column after column. The C language on the other hand, follows a row-major order, which means that the data is stored row after row. In the case of a rectangular array, like an augmented image array would be, this can be dealt with simply by switching indices. Thus, to get at the FORTRAN element `array(i,j)` in the C routine, use `array[j-1][i-1]`. Subtracting one from the indices in C is necessary because FORTRAN by default starts arrays at the one position, and C by default starts at the zero position.

Combining the code written to interface with the SEP program with the code for the SEP program results in a complete C and FORTRAN callable image display package. The next step then is to describe how to use the SEP program. This is the subject of the next chapter. For a thorough treatment of the fork and pipe UNIX systems calls, reference either Mikes [4], Rochkind [9], or Stevens [11] and on-line documentation.

CHAPTER IV

USING THE SEP PROGRAM

Introduction

The SEP program is available as a stand alone application, and as a library which can be called from both the C and FORTRAN languages.

This user's guide is organized into three major sections. The first section describes how to execute the program using either the SEP library or the stand alone application. The second section is a description of how to use the tools provided in SEP. And the third section covers conversion of existing images to the SEP format, and how to invert an image in the SEP format.

The first section includes example program code using the SEP library for both the C and FORTRAN languages, as well as compiling procedures. The third section includes an example program for converting the pix image file format (DeAnza format) to the SEP image file format. The third section also includes an example program for inverting the colormap of an SEP image. The SEP image file format is described in the second section.

First a note of caution: SEP is an X Windows based program, and as such cannot be successfully called from any windowing environment that does not support X Windows. The only way to call the program on the Icaen Apollo network, for example, is from the X Windows display manager. (On Icaen you must have started X Windows before you call this program, either with xllstart or login to a vue server)

Getting the SEP Program Running

Starting the Stand Alone Version

The SEP program can be utilized as either a stand alone application or as a subroutine called from either a C or FORTRAN language program. To use the program as a stand alone application the user must either have access to the SEP executable code, or all the source code for the program.

To use the stand alone version, the user simply enters the command 'sep'. The executable code must either be in the directory from which it is to be called or the path to the executable must be in the shell resource file. The stand alone can be run with or without a file to be displayed. To run it with a file to be displayed, simply include the filename on the command line. For example, "sep filename". Note: this version can also be run as a background process with the "&" command.

To use the stand alone program from its source code, first ensure that all parts are present. This can be done by

checking the make file objects list. When all parts are present in the same directory simply enter 'make'. When make gives its final message "Done sep functional" simply execute the program as above.

Starting SEP from the FORTRAN Language

To use SEP from a FORTRAN program you must have a FORTRAN program and access to the SEP program library described below. In the FORTRAN program the user simply calls SEP as if it were any other subroutine. For the FORTRAN language, the SEP call is to fsep. The fsep routine is a subroutine that requires three arguments. They are, in order: the array that contains the image data; the x direction size (width); and the y direction size (height). The proper syntax to call fsep is shown below in Figure 10.

```
call fsep( image, width, height )
```

Figure 10: FORTRAN Syntax to Call the SEP Program

The width and height variables must be declared as integer, or integer*4, and the image array variable must be declared as integer*2. The user should also be aware that the maximum array size is 1024 by 1024. Larger arrays will cause a segmentation fault or will not pass properly to the current version of the SEP program. The image array should be declared as 'name(1024, 1024),' and not as 'name(0:1024,

0:1024)' (using (0:1023,0:1023) would be acceptable under certain circumstances). Because of the required interface to C, starting the array at 0 would result in an error and would not accurately pass the image array data to the SEP program.

The user is cautioned to ensure that image array values do not exceed 255. The range expected by SEP is 0 to 255, and values greater or less than this will yield strange image display results or possibly a segmentation fault if the pixel values are large enough.

To compile a FORTRAN program that calls the fsep routine, you must include the sep_m68k.a library. An example of a compile command, including the sep_m68k.a library is shown in Figure 11, below.

```
f77 myprog.f sep_m68k.a -o myprog.out
```

Figure 11: FORTRAN Language Compile Command Example

You may either specify the path to the library in the compile call or copy the sep_m68k.a library to the directory that contains your program. In the example of Figure 11, it was assumed that the sep_m68k.a library had been copied to the directory containing the program, myprog.f. To specify the path to the library, in a UNIX environment, the user would do one of two things. The path can be specified by either including the path to the library in the shell resource file, or in the compile statement itself. For

example, if the library is in the -gdcarson/Public directory, the compile example in Figure 11 would be followed by -L -gdcarson/Public.

It is perhaps easiest to see how this works in an example program. An example of an FORTRAN program that calls the SEP program is included below in Figure 12.

```

      program fsample
*      Program to generate and display two images
*      Declare variables
      parameter(max=1024)
      integer*2  picture1(max,max),picture2(max,max)
      integer*4  w,h
*      Assign values to image array picture 1
      do i=1,512
        do j=1,512
          picture1(i,j)=(i/2.)-1
        enddo
      enddo
*      Assignment of width and height data
      w = 512
      h = 512
*      Call SEP display program
      call fsep ( picture1, w, h )
*      Create values for image array picture 2
      do i=1,512
        do j=1,512
          picture2(j,i)=(i/2.)-1
        enddo
      enddo
*      ready to call SEP again
      call fsep ( picture2, w, h )
*      That's it
      stop
      end

```

Figure 12: Example FORTRAN Language Program
 Using the SEP Program

From the example shown in Figure 12, it can be seen that executing the SEP program is very easy. In fact, in this

example the SEP program was started twice in two separate processes. A description of how to use the SEP functions follows the next section.

Starting SEP from the C Language

To use SEP from a C program the SEP call is to a routine named csep. As in the FORTRAN version the call to the SEP program from a C program is accomplished through a normal call to another C routine. The csep routine is a subroutine that requires three arguments. They are, in order: the array that contains the image data; the x direction size (width); and the y direction size (height). The proper syntax to call csep is shown in Figure 13.

```
csep( image, width, height );
```

Figure 13: C Language Syntax to Call the SEP Program

The width and height variables must be declared as int, and the image array variable must be declared as short int. The user should note the maximum array size and image array values mentioned above in the section starting SEP from the FORTRAN Language.

To compile a C program that calls the csep routine, you must include the sep_m68k.a library. An example of a compile

command for the C language which includes the sep_m68k.a library is shown in Figure 14.

```
cc myprog.c sep_c.a -o myprog.out
```

Figure 14: C Language Compile Command Example

```
#include <stdio.h>
#define BIG 1024
short int picture1[BIG][BIG], picture2[BIG][BIG];

main() {
    int x,y,width=512,height=512;
    for(x=0;x<width;x++){
        for(y=0;y<height;y++){
            picture1[x][y] = x/2;
        }
    }
    /* Call SEP */
    csep(picture1,width,height);
    /* Create second picture */
    for(x=0;x<width;x++){
        for(y=0;y<height;y++){
            picture2[y][x] = x/2;
        }
    }
    /* Call SEP again */
    csep(picture2,width,height);
    /* That's it */
}
```

Figure 15: Example C Language Program
Using the SEP Program

You may either specify the path to the library in the compiler call or copy the sep_m68k.a library to the directory that contains your program. The process for both of these cases is identical to those given for the FORTRAN language version above.

Again it is perhaps simplest to view an example of a C language program that makes use of the SEP program. An example of a C program that calls the SEP program is included below in Figure 15.

Here, again, it can be seen from the Figure that executing the SEP program is very easy. The use of SEP program itself is described in the next section.

Using SEP

The remainder of the information on the use of the SEP program is not language specific. The described functions operate the same whether SEP was started as a stand alone application or from another program.

As it can be seen in the examples (Figures 12 and 15 above) SEP can be called repeatedly from within the same program. However, since this is the case, the user is cautioned to ensure the SEP program is exited using the provided quit function. The SEP quit function is the third entry under the File menu item. To quit SEP you must place the mouse pointer on the "File" item in the menu bar (top left hand corner) and hold down the mouse button (mb1). Then simply drag the mouse down to the quit item in the list and release.

It should be noted that the only proper way to exit from the SEP program is through the use of the quit function. Both interrupt and kill have been tested and under normal

circumstances should provide proper exit, but that condition is not guaranteed. Therefore the user should always quit SEP using the quit function as described.

To use the save function, and the save as function, the method is very similar to the quit function. The user simply places the mouse pointer on the "File" item in the menu bar, holds down the mouse button (mb1), drags to either the save or save as item on the pull down menu and releases the mouse button. This action will result in an additional window popping up. This pop up window will contain a blank box, a "cancel" box, and an "OK" box. To enter the name you wish to give the file as it is saved, place the mouse pointer anywhere in the pop up window and type the name in on the keyboard. To begin the save operation the user must either click the mouse in the "OK" box on the pop up window or hit the return key with the mouse pointer in the pop up window. The save pop up window will remain up until the save operation has been completed. The user should also note that SEP will not allow any of its other functions to occur while it is saving.

Upon exiting from SEP, or searching the directory that SEP was called from the user will see two new files present after the save operation. The two files will be the saved image with the filename the user entered in the save window, and a second file with that filename with a .contour extension. The SEP program automatically adds an sep

extension to the provided file name. This is to ensure identification of SEP created files. For example, if the filename typed in was image, there will be two new files after the save operation. They would be 'image.sep' and 'image.contour'. The image file will be the complete contents of the image displayed at the time the save function was used. The contour file will be the number of pixels drawn upon and the x,y coordinate positions within the image. The file formats are described below in the next section.

To use the draw function, the method is a bit different. To draw on the displayed image place the mouse pointer on the displayed image and hold down a mouse button. The draw routine will then track the movement of the mouse pointer within the image and draw a green or white line along the path it followed. The line will only be green on a color monitor. This may be done as many times as desired. Each time the mouse button is released the drawing will stop, and each time the mouse button is again held down drawing will begin again from the present pointer position. If an error is made and the user wishes to remove the drawn line, simply place the mouse pointer on the "Erase" item on the menu bar, hold down the mouse button (mb1), drag the mouse to one of the functions listed, refer to Figure 16.

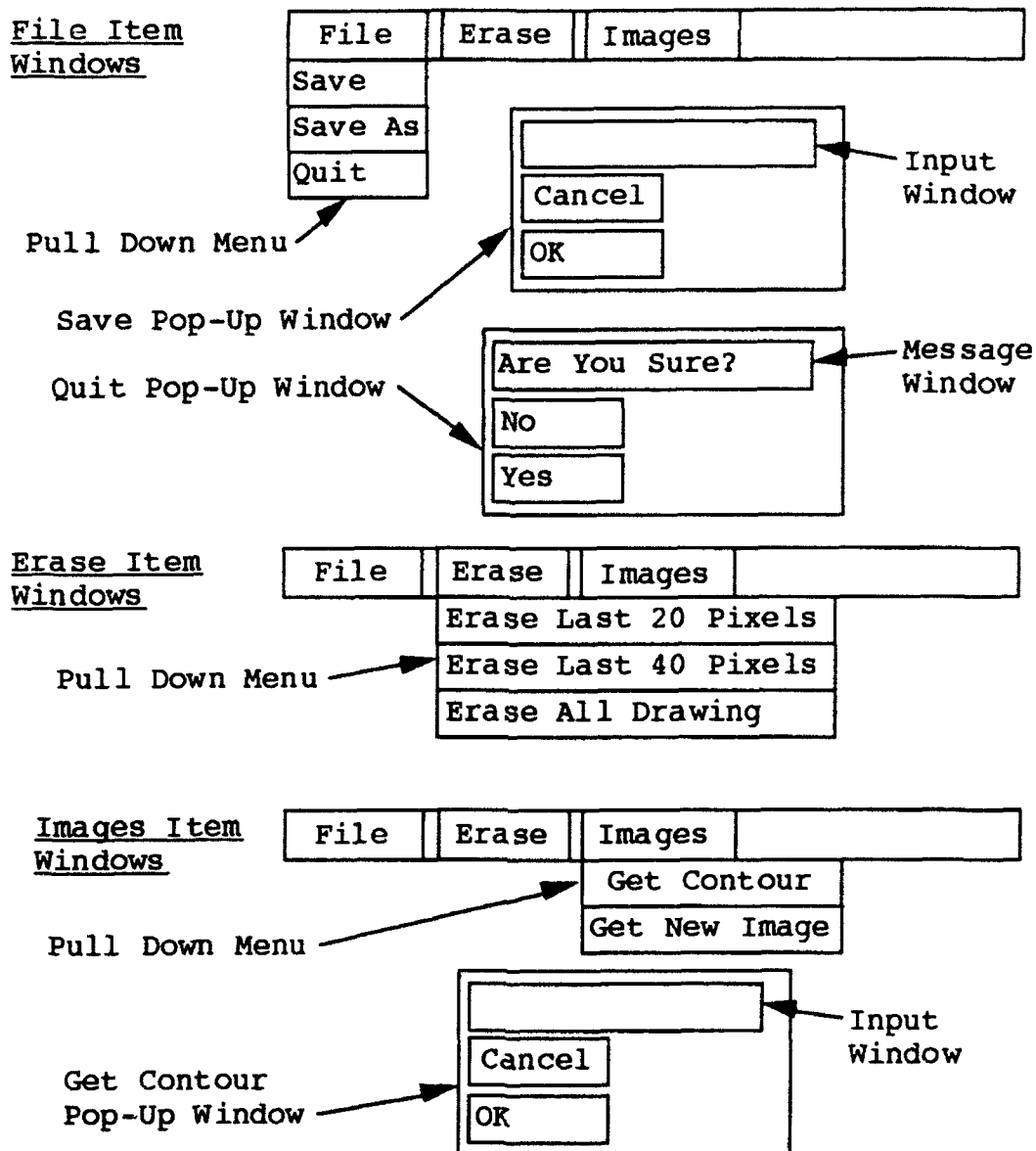


Figure 16: SEP Menu and Pop-Up Windows

Erasing of drawn lines can be done in 20 or 40 pixels increments depending on which menu item is chosen. All drawing can also be erased in this manner. The user should note that saved drawing cannot be erased.

To use the 'Images' function, once again the process is simple. Place the mouse pointer on the menu item 'Images' and hold down the mouse button (mbl). Drag the pointer to the 'Get Contour', or 'Get New Image' box and release. This action will cause a pop-up window to be displayed. The use of this pop-up window is identical to the use of the save functions pop-up window. Place the mouse in the window and type the name of the contour, or image file to be displayed. Note: the format of the contour file must be as described in the next section or an error will result.

The quit function, described in part above, also creates a pop-up window. Upon selection of the quit item from the file menu item, a window will be displayed. The window will display a message either "Are you Sure" or "Unsaved Drawing Exists Still Quit" and two boxes. The boxes contain "no" and "yes". Respond to the message by clicking either the "no" or "yes" box.

The SEP program is simple to use, and can display any image that can be put into the SEP image format. The last section describes how to convert one format to the SEP format.

Manipulating Images

The SEP Image Format

The image file format is simple. The entire image is written in binary format in single byte pieces. The first four bytes are the image width and height. To retrieve this data multiply the first and second bytes to get width and multiply the third and forth bytes to get the height. The remainder of the data is one byte per image pixel arranged in a stream of size width multiplied by height. The image-sep-contour file will be a listing of the x,y coordinates for all points within the image that were drawn on. The contour file format is also simple. The first line is the number of points in the file, and all of the following lines are x,y coordinate pairs for each point drawn in the order they were drawn from last to first. This file is in ASCII format. This is also the format that is required by the get contour function.

Converting to the SEP Image Format

As it has been mentioned, the image format for SEP is rather simple. Many images, however, are not formatted this way. Thus, it becomes necessary to convert images to the SEP format to use the program.

Figure 17 below contains a sample program to convert the DeAnza pix file format to the SEP file format.

```
#include <stddef.h>
#include <stdio.h>
typedef unsigned char DATA;
DATA image[1024][1024], coord[3];

main(){
    char pixfile[50], sepfile[50];
    FILE *fp1, *fp2;
    int x,y,width,height;
    long int here;

    printf( "Please enter the pix file to be converted.\n");
    gets(pixfile);
    if ((fp1 = fopen( pixfile, "rb" )) == NULL){
        printf( "Error opening file %s for input.\n", pixfile);
        return; }
    printf( "Please enter the sep filename.\n");
    gets(sepfile);
    if ((fp2 = fopen(strcat(sepfile, ".sep"), "wb"))==NULL){
        printf( "Error opening file %s for input.\n", sepfile);
        fclose( fp1 );
        return; }
    printf( "Please enter the width of the pix image.\n" );
    scanf( "%d", &width );
    printf( "Please enter the height of the pix image.\n" );
    scanf( "%d", &height );
    fread( image, sizeof (DATA), width * height, fp1 );
    coord[0] = coord[2] = 8;
    coord[1] = width/8;
    coord[3] = height/8;
    fwrite( coord, sizeof(DATA), 4, fp2 );
    fwrite( image, sizeof (DATA), width * height, fp2 );
    fclose( fp1 );
    fclose( fp2 );
}
```

Figure 17: Example Program to Convert
to SEP Image Format

Basically, the format to be converted needs to be read into memory and then written back out in the SEP format.

When using this example to create other conversion programs, the user is cautioned that the data used by the write statements to create the SEP file must be declared as unsigned char type.

There are many commonly used image data formats. Even in X programming there are many different image data formats. It is useful to mention the PBM program and its capabilities. The PBM program, Portable Bitmap program, is a collection of routines that convert between it's own format and many others. In PBM terms the SEP format would be a raw data format with a 4 byte header. Using this information the SEP format can be converted to and from Tagged Information File Format (TIFF), the Graphics Interchange Format (GIF), and many others, Refer to the PBM documentation for a complete description of all convertible formats.

Inverting the Colors of an SEP Image

In some cases it may be desirable to invert the colormap for an image. Inverting the colormap, switches the black and white values. In other words, the pixel value 0 becomes 255, 255 becomes 0, and all values in between are appropriately switched.

Figure 18 below, is an example program that inverts the colormap for an sep image file.

```

#include <stddef.h>
#include <stdio.h>
short int image_data[1024][1024];
typedef unsigned char DATA;
DATA stuff[1024*1024], coord[3];

main() {
    char imagefile[50], invfile[50];
    FILE *fp1, *fp2;
    int x,y,i,width,height;
    long int here;
    printf( "Please enter the input file name.\n");
    gets(imagefile);
    if ((fp1 = fopen( imagefile, "rb" )) == NULL) {
        printf( "Error opening file for input.\n");
        exit(); }
    printf( "Please enter an output file name if desired.\n");
    printf( "or hit return to overwrite original file.\n" );
    gets( invfile );
    if( invfile[0] == NULL ) { strcpy( invfile, imagefile); }
    else { strcat( invfile, ".sep" ); }
    fread( coord, sizeof (DATA), 4, fp1 );
    width = coord[0] * coord[1];
    height = coord[2] * coord[3];
    fread( stuff, sizeof (DATA), width * height, fp1 );
    /* convert input data into numerical data */
    here=0;
    for(x=0; x<width; x++)
        for(y=0; y<height; y++){
            here = here + 1;
            image_data[x][y] = stuff[here]; }
    fclose( fp1 );
    /* now invert the colors of the data read */
    for(x=0; x<width; x++)
        for(y=0; y<height; y++) {
            image_data[x][y] = -1*image_data[x][y];
            image_data[x][y] = image_data[x][y] + 255; }
    /* now put data back into the writable array */
    here = 0;
    for(x=0; x<width; x++)
        for(y=0; y<height; y++){
            here = here + 1;
            stuff[here] = image_data[x][y]; }
    if ((fp2 = fopen( invfile, "wb" )) == NULL) {
        printf( "Error opening file %s for input.\n", invfile);
        exit(); }
    fwrite( coord, sizeof (DATA), 4, fp2 );
    fwrite( stuff, sizeof (DATA), width * height, fp2 );
}

```

Figure 18: Example Program to Invert Colormap

The program shown, reference Figure 18, simply reads the image data file, changes each value to a negative of itself, and adds 255 to this value. This process will invert the color scale for the read image file.

Getting Hard Copy

There are numerous ways to obtain hard copy of an SEP displayed image. Almost always all such methods will be system dependent. On the Icaen network the best means of obtaining a print out is through the use of the shareware program XV. With the SEP program running, use the grasp function of XV to grab the image as desired and then define its output. On Icaen the output should be defined as regular PostScript. This file can then be printed on any PostScript printer.

Starting and using SEP is fairly straight forward, as is converting to the SEP image format, and inverting SEP files. The one remaining task to accomplish in this paper then is a discussion of the validation of the SEP program. This is the subject of the next chapter.

CHAPTER V

TESTING AND EVALUATION OF SEP

Testing and evaluation of the SEP program was conducted in two phases. The first phase was conducted to verify the accuracy of the program and its abilities. The second phase was actual use of the program by students from the College of Engineering.

During the first phase of testing, the program was provided a wide range of image sizes. The purpose of this was to ensure the program could accept and accurately manipulate images of all sizes through out its range. As stated earlier, the program's maximum image size is 1024 by 1024 pixels. Therefore, images ranging from 1 by 1 pixel through the maximum size 1024 by 1024 pixels were input to the program from both C and FORTRAN programs. Obviously not every conceivable size was input, but the entire range was tested. Table 1 contains the list of image sizes input to the program.

Table 1: Test Image Sizes and Results

Image Size (pixels)	Output Size (pixels)
1 x 1	1 x 1
32 x 32	32 x 32
64 x 64	64 x 64
128 x 128	128 x 128
128 x 256	128 x 256
128 x 512	128 x 512
256 x 256	256 x 256
256 x 512	256 x 512
256 x 1024	256 x 1024
512 x 512	512 x 512
768 x 1024	768 x 1024
1024 x 1024	1024 x 1024

To determine the size of the displayed image, the input image was saved to a file by the program. A separate program was written to read the saved file and count the number of pixels in width and height across the saved file. Table 1 shows the results of reading the SEP saved image file. In every case the program was 100% accurate.

In each test image shown in Table 1, a gray scale was generated by both a C and FORTRAN program. In images of less than 256 on one side, the gray scale was divided evenly in bars running the length at the shortest side. This was done

to ensure the full gray scale was reproduced by the colormap of the program. This was again verified by reading a file saved by the SEP program. In this examination, the file saved by the SEP program was compared to the original image data. Table 2 shows the results of this comparison.

Table 2: Test Image Pixel Comparisons

image size (pixels)	number of pixels in image	number of matches between saved file and original
32 x 32	1024	1024
64 x 64	4096	4096
128 x 128	16384	16384
128 x 256	32768	32768
128 x 512	65536	65536
256 x 256	65536	65536
256 x 512	131072	131072
256 x 1024	262144	262144
512 x 512	262144	262144
768 x 1024	786432	786432
1024 x 1024	1048576	1048576

The pixel comparison was conducted by comparing each pixel from the saved image with the corresponding pixel

originally generated. As it can be seen in Table 2, every image was a 100% match for the original data.

The last part of the first phase examined the draw function. A program was written to generate an all white image with a black square in the center. The image was 512 x 512 pixels in size, and the black square was 100 pixels on a side centered in the white image. The white image with a black square provided an easily discernible location within the image. The location of the transition from white to black was therefore known *a priori*. This was done to provide a clearly identifiable location on the image where a line could be drawn.

As it turned out the most difficult part of this test was drawing a straight line with the mouse pointer. To perform the test a line was drawn 100 pixels in length along the side of the black square. Once this was accomplished the image with the line was saved by the program. This saved file was then read and compared to the known location of the edge of the black square. The correlation between the known edge and the drawn line was 1 to 1. In other words, the drawing function was verified to be 100% accurate.

Having demonstrated the accuracy of the display and drawing functions it was time to begin the second phase. The second phase involved providing the program libraries and sample programs to several students for testing. These students were then asked to compile and run the sample

program, as well as generate their own images and use the program. Some students chose to write programs to read images and display them using the SEP program. All students were asked to use all of the functions of the program, and report all results.

The results of compiling and running the sample program were excellent. All students who attempted this were able to do it without error. The sample program ran and displayed the image as expected.

The results of the students either creating their own images or reading in an image and displaying it showed many difficulties. Fortunately, none of these problems was the result of an error in the SEP program. Typically the errors involved image data points greater than 255. As was mentioned in chapter 4, this can cause strange display results. This portion of the testing certainly validated this point.

Overall, the program performed exactly as expected. The programs use ability, accuracy, and functionality was validated. Thereby proving the program had met the stated requirements, and was capable of performing the required tasks.

CONCLUSION

Completing the Package

After the SEP program was completed and verified to be functioning as required, several additional routines were added to 'complete' the package. These routines were not part of the project as indicated in this paper, rather they are additions to make the SEP program a more complete image processing package.

The first of these routines is a read utility. This program was written to be C, and FORTRAN compatible. The read utility simply reads image files that are in the SEP image format. This routine returns a short int array with the image data in it, and size array with x and y dimension information. Use of this program is identical to the others described in this paper.

The second of these routines is a write utility. This program was also written to be C and FORTRAN compatible. The write utility is basically the opposite of the read utility. It is passed a short int image array, and the size information, and it writes the image data to file in the SEP image format. Again, use of this routine is identical to the others in this paper.

These two routines, the read and write utilities, can then be used by image processing programmers to deal with SEP image files and stay removed from the details of image file format.

Enhancing SEP

The SEP program could be used to create an image manipulation program. Additions to the SEP program would be fairly straight forward. Additional functions could be added to the menu bar function by adding additional items to the pull down menus or adding new pull down menus. Some ideas for additions to the program include the following:

The ability to annotate the image with text. This could be accomplished by simply drawing a string on the image or draw windows much like the draw function puts the cursor position information on the SEP parent window.

The ability to enhance the image, such as filtering, colormap inversion, interpolation and decimation. These functions could be written normally and then called to act upon the displayed image in a fashion similar to the erase function.

The ability to display pseudo color or color images. This ability could be added by restructuring the colormap to include more colors and altering how pixel values are interpreted by the colormap array variable.

The ability to save the drawn contour only. This ability could be very easily added to the existing program by a simple change to the Save As function. This function could be renamed Save Contour, and the associated subroutine could be amended to save any drawn contour data to file.

The alteration of SEP to become an image manipulation program is wide open area of possibilities. With the appropriate conversion routines added to the menu, it could even be programmed to display other image formats.

REFERENCES

1. Barkakati, N., X Window System Programming, SAMS, Carmel, Ind., 1991.
2. Darnel, P.A., Margolis, P.E., C A Software Engineering Approach, Springer-Verlag, New York, 1991.
3. Gonzalez, R. C., Wintz, P., Digital Image Processing, Addison-Wesley, Reading, Mass., 1987.
4. Mikes, S., UNIX for MS-DOS Programmers, Addison-Wesley Pub. Co., Reading, Mass., 1989.
5. Nye, A., Xlib Programming Manual : for Version 11 of the X Window System, O'Reilly & Associates, Sebastopol, California, 1991.
6. Nye, A., Xlib Reference Manual : for Version 11 of the X Window System, O'Reilly & Associates, Sebastopol, California, 1991.
7. Nye, A., X Toolkit Intrinsics Programming Manual Edition for X11, Release 4, O'Reilly & Associates, Sebastopol, California, 1990.
8. Nye, A., X Toolkit Intrinsics Reference Manual, O'Reilly & Associates, Sebastopol, California, 1991.
9. Rochkind, M. J., Advanced UNIX Programming, Prentice-Hall, Englewood Cliffs, N.J., 1985.
10. Scheifler, R. W., X Window System : The Complete Reference to Xlib, X Protocol, Digital Press, Bedford, MA. 1990.
11. Stevens, W. R., UNIX Network Programming, Prentice-Hall, Englewood Cliffs, N.J., 1990.
12. Tondo, C. L., Mastering MAKE : A Guide to Building Programs on DOS and UNIX Systems, Prentice Hall, Englewood Cliffs, N.J., 1992.

13. Young, D. A., The X Window System : Programming and Applications with Xt, Prentice Hall, Englewood Cliffs, N.J., 1990.